```
/*==========================================================================
Copyright July 2003 4G Color
All rights reserved
GD 7.18.03
==========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define inputFILENAME  "4gFilterForm"
#define outputFILENAME "4gParamArray"

typedef unsigned char U8 ;
typedef short   UVAL;       // uval is typically (-100,100)

enum colorDef { red, yellow, green, cyan, blue, magenta, all};
const int nColors = 7;
const int versionCode = 3314;

// filter basics ============================================================
char * filterStrings[] = {
    "     red yellow green  cyan  blue magenta all                    (+100,      -100)",//  0
    "  bright colors        (moreVivid, whiter)",                 //  1
    "  deep colors          (moreVivid, blacker)",                //  2
    "  midtone colors       (moreVivid, grayer)",                 //  3
    "  midtone lightness    (lighter,   darker)",                 //  4
    "  whitePoint           (positive,  negative)",               //  5
    "  lightGrayPoint       (positive,  negative)",               //  6
    "  darkGrayPoint        (positive,  negative)",               //  7
    "  blackPoint           (positive,  negative)",               //  8
    "  color shift          (>rygcbm,   <rygcbm)",                //  9
    "  light gray neutrals  (whiter,    grayer)",                 //  10
    "  dark gray neutrals   (blacker,   grayer)",                 //  11
    "  textures             (sharper,   smoother)",               //  12
    "  edges                (sharper,   smoother)"                //  13
};


// filter structure, size is 2*((9*6)+4) = 116 bytes
typedef struct
{
    // neutral axis (NP) color correction
    UVAL    whiteNP         [nColors]; // (+,-)
    UVAL    lightGrayNP     [nColors]; // (+,-)
    UVAL    darkGrayNP      [nColors]; // (+,-)
    UVAL    blackNP         [nColors]; // (+,-)

    // neutral brightness correction
    UVAL    lightGray;                 // (whiter,grayer)
    UVAL    darkGray;                  // (blacker,grayer)

    // saturated color correction
    UVAL    colorShift      [nColors]; // (r>y>g>c>b>m>r,m<r<y<g<c<b<m)

    // tone correction
    UVAL    brightColors    [nColors]; // (moreVivid,whiter)
    UVAL    deepColors      [nColors]; // (moreVivid,blacker)
    UVAL    midToneColor    [nColors]; // (moreVivid,grayer)
    UVAL    midToneLightness[nColors]; // (lighter,darker)

    // acuity
    UVAL    texture;                   // (sharper,smoother)
    UVAL    edges;                     // (sharper,smoother)
} filterDEF;
```

ATTACHMENT 8

```
void writeFilterForm(FILE *pForm, filterDEF fx)
{
    int i;
    fprintf( pForm, "%s\n",     filterStrings[0]);      // red yellow green  cyan  blue magent

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.brightColors[i]  );
        fprintf( pForm, "%s\n", filterStrings[1]);      // (moreVivid,whiter)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.deepColors[i]    );
        fprintf( pForm, "%s\n", filterStrings[2]);      // (moreVivid,blacker)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.midToneColor[i]  );
        fprintf( pForm, "%s\n", filterStrings[3]);      // (moreVivid,grayer)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.midToneLightness[i]  );
        fprintf( pForm, "%s\n", filterStrings[4]);      // (lighter,darker)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.whiteNP[i]   );
        fprintf( pForm, "%s\n", filterStrings[5]);      // (+,-)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.lightGrayNP[i]   );
        fprintf( pForm, "%s\n", filterStrings[6]);      // (+,-)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.darkGrayNP[i]    );
        fprintf( pForm, "%s\n", filterStrings[7]);      // (+,-)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.blackNP[i]   );
        fprintf( pForm, "%s\n", filterStrings[8]);      // (+,-)

    for(i=0; i<7; i++)
        fprintf( pForm, "%6d",  fx.colorShift[i]    );
        fprintf( pForm, "%s\n", filterStrings[9]);      // (lighter,darker)

    fprintf( pForm, "%42d%s\n",  fx.lightGray,  filterStrings[10]);
    fprintf( pForm, "%42d%s\n",  fx.darkGray,  filterStrings[11]);
    fprintf( pForm, "%42d%s\n",  fx.texture,  filterStrings[12]);
    fprintf( pForm, "%42d%s\n",  fx.edges,  filterStrings[13]);
}

// scan the user text form
void scanFilterForm(FILE *pForm, filterDEF *fx);
void scanFilterForm(FILE *pForm, filterDEF *fx)
{
    // we just scan uninteresting text with this buffer
    int i, temp;
    char buffer[100];
    fgets( buffer, 100, pForm );                        // red yellow green  cyan  blue magent

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> brightColors[i] = temp;}
        fgets( buffer, 100, pForm );                    // (moreVivid,whiter)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> deepColors[i] = temp;}
        fgets( buffer, 100, pForm );                    // (moreVivid,blacker)
```

```
    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> midToneColor[i] = temp;}
        fgets( buffer, 100, pForm );                    // (moreVivid,grayer)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> midToneLightness[i] = temp;}
        fgets( buffer, 100, pForm );                    // (lighter,darker)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> whiteNP[i] = temp;}
        fgets( buffer, 100, pForm );                    // (+,-)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> lightGrayNP[i] = temp;}
        fgets( buffer, 100, pForm );                    // (+,-)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> darkGrayNP[i] = temp;}
        fgets( buffer, 100, pForm );                    // (+,-)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx-> blackNP[i] = temp;}
        fgets( buffer, 100, pForm );                    // (+,-)

    for(i=0; i<7; i++)
    {   fscanf( pForm, "%d", &temp);    fx->colorShift[i] = temp;}
        fgets( buffer, 100, pForm );                    // (lighter,darker)

    fscanf( pForm, "%d", &temp);    fx-> lightGray = temp;  fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> darkGray = temp;   fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> texture = temp;    fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> edges = temp;      fgets( buffer, 100, pForm );
}


// this is initialized sideways
void initFilter(filterDEF *fx);
void initFilter(filterDEF *fx)
{
    for(int i=0; i<nColors; i++)
    {
        fx-> whiteNP            [i] = 0;
        fx-> lightGrayNP        [i] = 0;
        fx-> darkGrayNP         [i] = 0;
        fx-> blackNP            [i] = 0;
        fx-> colorShift         [i] = 0;
        fx-> brightColors       [i] = 0;
        fx-> deepColors         [i] = 0;
        fx-> midToneColor       [i] = 0;
        fx-> midToneLightness   [i] = 0;
    }
    fx-> lightGray  = 0;
    fx-> darkGray   = 0;
    fx-> texture    = 0;
    fx-> edges      = 0;
}

void printFilter(filterDEF fp);
void printFilter(filterDEF fp)
{
    printf("The size of the filter is %5d bytes.\n", sizeof(fp)); //fa.lightGray);
    for(int i=0; i<nColors; i++)
```

```
    {
        printf("%5d", fp.whiteNP[i]);
        printf("%5d", fp.lightGrayNP[i]);
        printf("%5d", fp.darkGrayNP[i]);
        printf("%5d", fp.blackNP[i]);
        printf("%5d", fp.colorShift[i]);
        printf("%5d", fp.brightColors[i]);
        printf("%5d", fp.deepColors[i]);
        printf("%5d", fp.midToneColor[i]);
        printf("%5d\n", fp.midToneLightness[i]);
    }
    printf("%5d\n", fp.lightGray);
    printf("%5d\n", fp.darkGray);
    printf("%5d\n", fp.texture);
    printf("%5d\n", fp.edges);
}
// end filter basics ============================================================

// Fog   Filter ================================================================
char * fogStrings[] = {
    "   TRANSFORM CREATOR (Compiles a compact image transform.)",                    //  0
    "Resize, Reshape, and Crop",                                                     //  1
    "   input pixels",                                                               //  2
    "   input lines",                                                                //  3
    "   output pixels",                                                              //  4
    "   output lines",                                                               //  5
    "   cropSelect              (inscribe, superscribe, anamorphic)",                //  6
    "Exposure and Coding Correction",                                                //  7
    "   auto Exposure           (autoExposureOff, autoExposure100, autoExposure80)", //  8
    "   auto Color Balance      (autoColorBalanceOff, autoColorBalanceOn)",          //  9
    "   jpeg Filter             (jpegOff, jpegOn)",                                  //  10
    "Filter Selection and Composition",                                              //  11
    "   filter select           (fA, fB, A+B, ABblend)",                             //  12
    "   filter A gain           (+,-)",                                              //  13
    "   filter B gain           (+,-)",                                              //  14
    "Forground Filter--filter A",                                                    //  15
    "Background Filter--filter B",                                                   //  16
    "   Version code. Copyright 2003 by 4G Color. All rights reserved."              //  17
};

typedef enum   {inscribe, superscribe, anamorphic}                    cropDEF ;
typedef enum   {fA, fB, AplusB, ABblend}                              compositionDEF;
typedef enum   {autoExposureOff, autoExposure100, autoExposure80}     autoExposureDEF;
typedef enum   {autoColorBalanceOff, autoColorBalanceOn}              autoColorBalanceDEF;
typedef enum   {jpegOff, jpegOn}                                      jpegFilterDEF;

// process structure
typedef struct
{
    // resize, reshape, and crop
    long    inPixels;
    long    inLines;
    long    outPixels;
    long    outLines;
    cropDEF cropSelect; // (inscribe, superscribe, anamorphic)

    // compensation
    autoExposureDEF     autoExposureSelect;     // {autoExposureOff, autoExposure100, autoExp
    autoColorBalanceDEF autoColorBalanceSelect; // {autoColorBalanceOff, autoColorBalanceOn}
    jpegFilterDEF       jpegFilterSelect;       // {jpegOff, jpegOn}

    // filter composition and definition
    compositionDEF  filterSelect;       // {fA, fB, AplusB, ABblend}
    short           filterAgain;        // (+,-)
```

```
    short           filterBgain;        // (+,-)
    filterDEF       filterA;
    filterDEF       filterB;
} fogParamArrayDEF;

void writeFogForm(FILE *pForm, fogParamArrayDEF fx)
{
    int temp = fx.filterAgain;
    fprintf( pForm, "%s\n",fogStrings[0]);  // TRANSFORM CREATOR
    fprintf( pForm, "%s\n",fogStrings[1]);  // resize, reshape, and crop",
    fprintf( pForm, "%6d%s\n", fx.inPixels, fogStrings[2]);
    fprintf( pForm, "%6d%s\n", fx.inLines, fogStrings[3]);
    fprintf( pForm, "%6d%s\n", fx.outPixels, fogStrings[4]);
    fprintf( pForm, "%6d%s\n", fx.outLines, fogStrings[5]);
    fprintf( pForm, "%6d%s\n", fx.cropSelect, fogStrings[6]);
    fprintf( pForm, "%s\n",fogStrings[7]);  // compensation
    fprintf( pForm, "%6d%s\n", fx.autoExposureSelect, fogStrings[8]);
    fprintf( pForm, "%6d%s\n", fx.autoColorBalanceSelect, fogStrings[9]);
    fprintf( pForm, "%6d%s\n", fx.jpegFilterSelect, fogStrings[10]);
    fprintf( pForm, "%s\n",fogStrings[11]); // filter composition and definition
    fprintf( pForm, "%6d%s\n", fx.filterSelect, fogStrings[12]);
    fprintf( pForm, "%6d%s\n", fx.filterAgain, fogStrings[13]);
    fprintf( pForm, "%6d%s\n", fx.filterBgain, fogStrings[14]);

    fprintf( pForm, "%s\n",fogStrings[15]); // filter A
    writeFilterForm(pForm, fx.filterA);

    fprintf( pForm, "%s\n",fogStrings[16]); // filter B
    writeFilterForm(pForm, fx.filterB);

    fprintf( pForm, "%6d%s\n", versionCode, fogStrings[17]);
}

// scan the fogForm
int scanFogForm(FILE *pForm, fogParamArrayDEF *fx);
int scanFogForm(FILE *pForm, fogParamArrayDEF *fx)
{
    int temp;
    char buffer[100];

    fgets( buffer, 100, pForm );    // TRANSFORM CREATOR
    fgets( buffer, 100, pForm );    // resize, reshape, and crop",
    fscanf( pForm, "%d", &temp);    fx-> inPixels = temp;   fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> inLines = temp;    fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> outPixels = temp;  fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> outLines = temp;   fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> cropSelect = (cropDEF)temp;    fgets( buffer, 100, pF
    fgets( buffer, 100, pForm );    // compensation
    fscanf( pForm, "%d", &temp);    fx-> autoExposureSelect = (autoExposureDEF)temp;    fgets(
    fscanf( pForm, "%d", &temp);    fx-> autoColorBalanceSelect = (autoColorBalanceDEF)temp;
    fscanf( pForm, "%d", &temp);    fx-> jpegFilterSelect = (jpegFilterDEF)temp;    fgets( buf
    fgets( buffer, 100, pForm );    // filter composition and definition
    fscanf( pForm, "%d", &temp);    fx-> filterSelect = (compositionDEF)temp;   fgets( buffer,
    fscanf( pForm, "%d", &temp);    fx-> filterAgain = temp;    fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &temp);    fx-> filterBgain = temp;    fgets( buffer, 100, pForm );

    fgets( buffer, 100, pForm );    // filter A
    scanFilterForm(pForm, &fx->filterA);

    fgets( buffer, 100, pForm );    // filter B
    scanFilterForm(pForm, &fx->filterB);

    fscanf( pForm, "%d", &temp, fogStrings[17]);
    return temp;
```

```
}

void initFog(fogParamArrayDEF *pax);
void initFog(fogParamArrayDEF *pax)
{
    // resize, reshape, and crop
    pax-> inPixels      = 0;
    pax-> inLines       = 0;
    pax-> outPixels     = 0;
    pax-> outLines      = 0;
    pax-> cropSelect    = inscribe;

    // compensation
    pax-> autoExposureSelect        = autoExposureOff;
    pax-> autoColorBalanceSelect    = autoColorBalanceOff;
    pax-> jpegFilterSelect          = jpegOff;

    // filter composition and definition
    pax-> filterSelect  = fA;
    pax-> filterAgain   = 100;
    pax-> filterBgain   = 100;

    initFilter( &(pax->filterA) );
    initFilter( &(pax->filterB) );
}

void printFog(fogParamArrayDEF pa);
void printFog(fogParamArrayDEF pa)
{
    printf("%5d\n", pa.inPixels);
    printf("%5d\n", pa.inLines);
    printf("%5d\n", pa.outPixels);
    printf("%5d\n", pa.outLines);
    printf("%5d\n", pa.cropSelect);

    printf("%5d\n", pa.autoExposureSelect);
    printf("%5d\n", pa.autoColorBalanceSelect);
    printf("%5d\n", pa.jpegFilterSelect);

    printf("%5d\n", pa.filterSelect);
    printf("%5d\n", pa.filterAgain);
    printf("%5d\n", pa.filterBgain);

    printFilter( pa.filterA);
    printFilter( pa.filterB);

}
// End Fog   Filter ============================================================

// read this
int main()
{
    fogParamArrayDEF xxx;
    FILE * pForm;

    pForm = fopen (outputFILENAME, "rb");
    if( pForm==NULL ) goto PAUSE;
    fread( &xxx, sizeof(xxx), 1, pForm );
    fclose(pForm);

    pForm = fopen( "testForm", "w" );
    if( pForm==NULL ) goto PAUSE;
    writeFogForm(pForm, xxx );
    fclose(pForm);
```

```
return 0;
PAUSE:
    printf("\n      Error. (Enter a number to exit.)\n");
    int temp;
    scanf("%d", &temp);
    printf("\n      ... bye\n");

}

#if 0
//  ********************************************************************************************
// user interface --> plist --> ppf translations and control

// text form layout                                                    user variable names
char * pStrings[] = {
    "                        TRANSFORM CREATOR (Compiles a compact image transform.)", //  0 ..
    "  Digital Camera Auto-Exposure (off,on100,on80,on50,declip)",  //  1   autoExposure
    "  Select foreground            (off,bot,left,right,cent,centBot)", //  2   foreground
    "  Compose                      (A,B,A+B,A/B) ",            //  3   compose
    "  Resize   (pixelsIn,linesIn,pixelsOut,linesOut)", //  4   reSize[4]
    "  Resize Format                (inscribe,superscribe,anamorphic)", //  5   format
    "  Image File Format            (rgb.raw,rgbPlaner.raw,cmyk.raw,cmykPlaner.raw)",   //  6
    "  Compression Compensation     (off,data,display,jpeg,)",  //  7   diags
    "       -------------   filter A   -------------",          //  8   ...
    "    red yellow green  cyan  blue magenta all                    (+100, -100)",//  9 ...
    "  brightColors         (moreVivid, whiter)",                 //  10  uVal[7][0]
    "  deepColors           (moreVivid, blacker)",                //  11  uVal[7][1]
    "  midToneColor         (moreVivid, grayer)",                 //  12  uVal[7][2]
    "  midToneLightness     (lighter,   darker)",                 //  13  uVal[7][3]
    "  colorShift           (r>y>g>c>b>m,  r<y<g<c<b<m)",         //  14  uVal[7][4]
    "  whitePoint           (positive,  negative)",               //  15  uVal[7][5]
    "  lightGrayNeutrals    (whiter,    grayer)",                 //  16  lGray
    "  darkGrayNeutrals     (blacker,   grayer)",                 //  17  dGray
    "  acuity               (sharper,   smoother)",               //  18  acuity
    "  filter A gain        (positive,  negative)",               //  19  fGain
    "       -------------   filter B   -------------",          //  20  ...
    "    red yellow green  cyan  blue magenta all                    (+100, -100)",//  21    ..
    "  brightColors         (moreVivid, whiter)",                 //  22  uVal[7][0]
    "  deepColors           (moreVivid, blacker)",                //  23  uVal[7][1]
    "  midToneColor         (moreVivid, grayer)",                 //  24  uVal[7][2]
    "  midToneLightness     (lighter,   darker)",                 //  25  uVal[7][3]
    "  colorShift           (r>y>g>c>b>m,  r<y<g<c<b<m)",         //  26  uVal[7][4]
    "  whitePoint           (positive,  negative)",               //  27  uVal[7][5]
    "  lightGrayNeutrals    (whiter,    grayer)",                 //  28  lGray
    "  darkGrayNeutrals     (blacker,   grayer)",                 //  29  dGray
    "  acuity               (sharper,   smoother)",               //  30  acuity
    "  filter B gain        (positive,  negative)",               //  31  fGain
    "  Version Code",                                             //  32  vCode
    "                       Copyright 2003 by 4G Color. All rights reserved." //  33    ...
};

// initialize user selected variables
void  image::initUserVals()
{
    // common variables
    autoExposure = 0;
    foreground = 0;
    compose = 0;
    reSize[0] = 0;  // if either width or height is 0, no change is applied
    reSize[1] = 0;
    reSize[2] = 0;
    reSize[3] = 0;
    format = 0;
```

```
    pack = 0;
    diags = 0;
    vCode = 133;

    // filter A values
    for(int i=0; i<7; i++)  for(int j=0; j<6; j++) uVal[i][j] = 0;
    lGray = 0;
    dGray = 0;
    acuity = 0;
    fGain = 100;

    // filter b values
    for(int i=0; i<7; i++)  for(int j=0; j<6; j++) uVal1[i][j] = 0;
    lGray1 = 0;
    dGray1 = 0;
    acuity1 =  0;
    fGain1 = 100;
}

// write or rewrite the user text form
void image::writeForm(FILE *pForm)
{
    fprintf( pForm, "%s\n",                                    pStrings[0]);   //  0   PROCESS DI
    fprintf( pForm, "%6d %s\n",     autoExposure,              pStrings[1]);   //  1   autoExposu
    fprintf( pForm, "%6d %s\n",     foreground,                pStrings[2]);   //  2   foreground
    fprintf( pForm, "%6d %s\n",     compose,                   pStrings[3]);   //  3   compose
    fprintf( pForm, "%6d %6d %6d %6d %s\n", reSize[0], reSize[1], reSize[2], reSize[3],     pS
    fprintf( pForm, "%6d %s\n",     format,                    pStrings[5]);   //  5   format
    fprintf( pForm, "%6d %s\n",     pack,                  pStrings[6]);   //  6   pack
    fprintf( pForm, "%6d %s\n",     diags,                     pStrings[7]);   //  7   diags
    fprintf( pForm, "%s\n",                                    pStrings[8]);   //  8   static fil
    fprintf( pForm, "%s\n",                                    pStrings[9]);   //  9   red yellow
    for(int j=0; j<6; j++)
    {
        for(int i=0; i<7; i++)  fprintf( pForm, "%6d", uVal[i][j]);         // 10-15
        fprintf( pForm, "%s\n",     pStrings[10+j]);
    }
    fprintf( pForm, "%42d %s\n",    lGray,                     pStrings[16]);  //  16  lGray
    fprintf( pForm, "%42d %s\n",    dGray,                     pStrings[17]);  //  17  dGray
    fprintf( pForm, "%42d %s\n",    acuity,                    pStrings[18]);  //  18   acuity
    fprintf( pForm, "%42d %s\n",    fGain,                     pStrings[19]);  //  19  fGain
    fprintf( pForm, "%s\n",                                    pStrings[20]);  //  20  static fil
    fprintf( pForm, "%s\n",                                    pStrings[21]);  //  21  red yellow
    for(int j=0; j<6; j++)
    {
        for(int i=0; i<7; i++)  fprintf( pForm, "%6d", uVal1[i][j]);            // 22-27
        fprintf( pForm, "%s\n",     pStrings[22+j]);
    }
    fprintf( pForm, "%42d %s\n",    lGray1,                    pStrings[28]);  //  28  lGray
    fprintf( pForm, "%42d %s\n",    dGray1,                    pStrings[29]);  //  29  dGray
    fprintf( pForm, "%42d %s\n",    acuity1,                   pStrings[30]);  //  30   acuity
    fprintf( pForm, "%42d %s\n",    fGain1,                    pStrings[31]);  //  31  fGain
    fprintf( pForm, "%42d %s\n",    vCode,                     pStrings[32]);  //  32  vCode
    fprintf( pForm, "\n%s\n",                                  pStrings[33]);  //  33  copyright
    fclose(pForm);
}

// scan the user text form
void image::scanForm(FILE *pForm)
{
    char buffer[100];
                                        fgets( buffer, 100, pForm );    //  0   ...
    fscanf( pForm, "%d", &autoExposure);fgets( buffer, 100, pForm );    //  1   autoExposu
    fscanf( pForm, "%d", &foreground);  fgets( buffer, 100, pForm );    //  2   statFilter
```

```
    fscanf( pForm, "%d", &compose);          fgets( buffer, 100, pForm );    //  3   rezEnhance
    fscanf( pForm, "%d", &reSize[0]);                                        //  4   reSize[2]
    fscanf( pForm, "%d", &reSize[1]);                                        //  4   reSize[2]
    fscanf( pForm, "%d", &reSize[2]);                                        //  4   reSize[2]
    fscanf( pForm, "%d", &reSize[3]);        fgets( buffer, 100, pForm );
    fscanf( pForm, "%d", &format);           fgets( buffer, 100, pForm );    //  5   format
    fscanf( pForm, "%d", &pack);             fgets( buffer, 100, pForm );    //  6   pack
    fscanf( pForm, "%d", &diags);            fgets( buffer, 100, pForm );    //  7   diags
                                             fgets( buffer, 100, pForm );    //  8   ...
                                             fgets( buffer, 100, pForm );    //  9   ...
    for(int j=0; j<6; j++)
    {
        for(int i=0; i<7; i++)  fscanf( pForm, "%d", &uVal[i][j]);           // 10-15
        fgets( buffer, 100, pForm );
    }
    fscanf( pForm, "%d", &lGray);            fgets( buffer, 100, pForm );    //  16  lGray
    fscanf( pForm, "%d", &dGray);            fgets( buffer, 100, pForm );    //  17  dGray
    fscanf( pForm, "%d", &acuity);           fgets( buffer, 100, pForm );    //  18  acuity
    fscanf( pForm, "%d", &fGain);            fgets( buffer, 100, pForm );    //  19  fGain
                                             fgets( buffer, 100, pForm );    //  20  ...
                                             fgets( buffer, 100, pForm );    //  21  ...
    for(int j=0; j<6; j++)
    {
        for(int i=0; i<7; i++)  fscanf( pForm, "%d", &uVal1[i][j]);          // 22-27
        fgets( buffer, 100, pForm );
    }
    fscanf( pForm, "%d", &lGray1);           fgets( buffer, 100, pForm );    //  28  lGray
    fscanf( pForm, "%d", &dGray1);           fgets( buffer, 100, pForm );    //  29  dGray
    fscanf( pForm, "%d", &acuity1);          fgets( buffer, 100, pForm );    //  30  acuity
    fscanf( pForm, "%d", &fGain1);           fgets( buffer, 100, pForm );    //  31  fGain
    fscanf( pForm, "%d", &vCode);            fgets( buffer, 100, pForm );    //  32  vCode

}


void  image::readProcess()
{
    FILE * pForm = fopen (FILENAME, "r");
    if( pForm==NULL )   // clear the process and write the form
    {
        pForm = fopen (FILENAME, "w");
        initUserVals();
        writeForm(pForm);
        fclose(pForm);
    }
    else                // scan and rewrite the form
    {
        scanForm( pForm );
        fclose(pForm);
        if( vCode != 133 ) initUserVals(); // the validity code is contaminated
        pForm = fopen (FILENAME, "w");
        writeForm(pForm);
        fclose(pForm);
    }
}

#endif

// copyright 2003, 4G Color, All rights reserved
```